ANALISIS Y DISEÑO DE ALGORITMOS

INFORME TRABAJO PRATICO ESPECIAL

SEGUNDA PARTE





INTEGRANTES: BOCCANFUSO LUCAS EZEQUIEL, BERTINO ARIEL EUGENIO

N° DE GRUPO: 36

AYUDANTE ASIGNADO: MENCHON MARTIN

E-MAIL: lucas.mdq.935@gmail.com, eugenioingenio10@gmail.com

INTRODUCCION AL PROBLEMA

En esta segunda parte del trabajo práctico especial de análisis y diseño de algoritmo entendimos que:

1. Tenemos una base de dato conformada por muchos libros en forma de archivos de texto.
2. Tenemos que de alguna forma llevar todos esos libros a una estructura en memoria secundaria.
3. Realizar búsquedas filtrando determinados libros en base a su/s genero/s.
4. Dichas búsquedas refinan el total de libros mediante servicios de refinación dejando o quitando ciertos libro de la solución final, y volvando esa información en un archivo final
5. Como minimo tenemos/podemos usar dos tipos de datos libro y un índice adaptándolo a nuestras necesidades obviamente respetando ciertas características.
6. Diseñar e implementar una aplicación simple de búsqueda de libros.
7. Aclaración como bibliografía usamos el libro de edebert schildt sugeriod por la catedra

ESPECIFICACION FORMAL DE TIPOS DE DATOS INVOLUCRADOS

TDA Libro

CLASS libro

IMPORTS natural, cadena

BASIC CONSTRUCTORS creaL

EFFECTIVE TYPE libro

OPERATIONS

crearL : cadena \* cadena \* natural \* cadena \* cadena \* cadena → libro;

obt\_T : libro → cadena;

obt\_A : libro → cadena;

obt\_P : libro → natural;

obt\_E : libro → natural;

obt\_G : libro → cadena;

obt\_C : libro → cadena;

mod\_T: libro \* cadena → libro;

mod\_A: libro \* cadena → libro;

mod\_P: libro \* cadena → libro;

mod\_E: libro \* cadena → libro;

mod\_G: libro \* cadena → libro;

mod\_C: libro \* cadena → libro;

AXIOMS … … … …

.

.

.

ENDO CLASS;

No utilizamos una estructura en particular, no vimos necesario para este caso utilizar una estructura en concreto simplemente variables de tipo cadena.

Asi mismo tampoco consideramos necesario hacer una explicación minuciosa de cada método simplemente que esta clase posee dos tipos de métodos los llamados getters por ejemplo: obt\_T() que obtiene el titulo en este caso asi también habrá uno para el genero, las paginas etc y por otro lado los denominados setter que modifican o dan un valor a estos campos por ejemplo mod\_G(string genero) que dado un genero (por parámetro) actualiza dicho campo especifico.

En lo que refiere a la complejidad temporal de los métodos anteriormente comentados hay que decir que de los getters solo obtemenos el valor de ese campo y de los setter análogamente los modificamos siendo todas operaciones que se comportan de manera constante en base a su complejidad temporal, la misma pertenece a O(1) para ambos tipos de métodos.

TDA INDICE

CLASS Indice

IMPORTS libro ,conjunto[libro]

BASIC CONSTRUCTORS crearIndice

EFFECTIVE TYPE Indice

OPERATIONS

crearIndice: → Indice;

agregarLibro : Indice \* libro → Indice;

devolverGenero : Indice \* cadena → conjunto[libro];

AXIOMS … … … …

.

.

.

END CLASS;

Para poder llevar un índice a memoria y disponer según sea necesario de el decidimos usar un árbol binario de búsqueda, en concreto cada nodo de este árbol es una estructura que posee los siguientes campos: genero(cadena), conjunto(de apuntadores a libros) y dos punteros izq y der que apuntan a otra estructura también con los mismos campos recién nombrados. Hacemos uso tanto de un conjunto desarrollado en la primera parte del trabajo practico especial como de libro especificado mas arriba.

Los métodos que posee este denominado índice son simplemente 3 detallados acontinuacion:

Indice(): simplemente inicializa un r cursor o puntero auxiliar para decir que se inicializo un variable de tipo índice y esta vacio. Realiza una operación de orden constante por lo tanto su complejidad es de O(1);

~Indice(): Recorre todo el índice borrarndo genero por genero al ser un ‘arbol’ índice hay que tener en cuenta los casos árbol degenerado (formato de lista) y el árbol ordenado balanceado completo, para el primer caso la complejidad temporal pertenece a O(n) el peor de los casos y para el segundo su complejidad ronda el O(log n), siendo en ambos casos n la cantidad de generos de índice,

agregarLibro(libro \* l): Este método toma todos los generos que posee una libro obviamente al principio es una cadena gigante con muchos generos separados por espacios en blanco, que va desmenuzado genero por genero agregando un puntero a este libro en el conjunto de libro que indica el genero recién obtenido, osea agrega tantas veces el libro a cada genero como generos tenga ese libro. Recorre el índice preguntando si el genero actual pertenece al índice si pertenece al índice agrega en el conjunto de punteros a libro un puntero apuntando a este libro de lo contrario hara lo mismo pero posterior a agregar un índice con ese genero nuevo siendo este el primer libro con ese genero.

devolverGenero(string genero, conjunto<libro\*> & resultado): Lleva a cabo lo siguiente, busca el genero que se quiere devolver si no lo encuentra crea un nodo con ese genero y obviamente no va a tener ningún libro en su conjunto de punteros a libros, luego agrega al conjunto resultado todos los libro que el genero a devolver posea, digamos que hace una copia por decirlo asi de todos los punteros a libros de ese genero, si justo no tiene libros el genero a devolver, lo informa por pantalla quedando el conjunto de punteros a libros de resultado vacio.

Aunque un poco intrincada la complejidad de este método se describe asi, en primer lugar busca y/o agrega en índice por lo tanto a lo suma recorre n nodos de índice, en segundo lugar, en ese nodo nuevo o no debe recorrer m cantidad de punteros de libros que posea el conjunto de dicho genero. Luego de manera teorica la complejida queda dada por O(n.m) pero cuando n y m tiende a un numero muy grande aveces se lo toma que tiene a infinito en ese punto dejan de distinguirse y se puede tomar para simplifica cuentas como que su complejidad resultante pertenece a O(n2).

A continuación detallaremos la complejidad temporal de los algoritmos que jugaron un papel en la solución en si del problema planteado.

procesarArchivos(const path & ruta, Indice &i, conjunto<libro\*> &todosLibros): función provista por la catedra se podría decir que recorre todos lo subdirectorios de un directorio dado buscado libros en este cosa archivos.txt, entonces, la complejidad esta dada por n cantidad de libros siendo esta O(n).

procesarInformacionLibro(const path & ruta, Indice &i, conjunto<libro\*> &todosLibros): nuevamente nos encontramos ante una función provista por la catedra pero en forma parcial ya que por si sola esta función lo único que hace es obtener de cada libro(osea cada archivo. txt), todos los campos que el mismo contiene autor, contenido etc. Entonces para hacer que encaje es algoritmo con lo pedido en la consigna es declara dentro de esta función una variable de tipo libro a la cual le iremos cargando campo a campo los valore extraidos del archivo. Luego de cargar el archivo a memoria quedando como un libro ese libro se agrega a un conjunto y ese conjunto se agrega a un nodo que se va a insertar en el índice con su genero correspondiente. La complejidad temporal se va regir por la compljidad propia de agregar un nodo al índice por lo tanto la complejidad pertenece a O(n) siendo n la cantidad de nodod del índice.

mostrarMenu(char & opcion): No hay mucho que explicar de esta función solo se encarga de mostrar una serie de opciones que en ese instante son solo palabras por pantalla o sea operaciones de orden constate, por lo tanto la complejidad temporal es de O(1).

mostrarSubMenu(char & opcion): muestra opciones por pantalla y pide una opción por teclado, operaciones constantes su complejidad es O(1).

pidoGeneroPorPantalla(): pide un genero por patalla y verifica que exista en el índice para poder luego procesar en funciones que invoquen a esta otra, todas las operaciones que realiza esta función son asignciones preguntas por lo tanto su complejidad es de O(1);

refinarBusqueda(conjunto<libro\*> &parcialLibros, Indice &i): este procedimiento se encarga de refinar las búsquedas dejando los resultados en un cojunto de resultados parciales que refina mediante sucivas llamados de los servicios, lo mas costoso temporalmente son las operaciones propias de conjunto con unión, intersección, etc, por lo tanto la complejidad de esta función es de O(n2).

mostrarResultado (conjunto<libro\*> &parcialLibros): muestra por pantalla todos los libros de un conjunto de punteros a libro por lo tanto a lo sumo recorre n cantidad de libros por lo tanto la complejidad temporal es de O(n).

guardarLibros(ofstream & arch, conjunto<libro\*> &parcialLibros): reinicia el cursor de parcialLibros para apunte al primero, y asi poder recorrerlo hasta la cantidad de elementos que tenga cargando al archivo la ruta, titulo y autor de cada libro, siendo n la cantdad de punteros a libro que tenga parcial libros la complejidad temporal es de O(n).

creoArchivo (conjunto<libro\*> &parcialLibros): pide una ruta por teclado que luego la completa con la extensión .txt para poder almacenar los libros de resultado parcial en un archivo de texto. Luego abre el archivo y carga todo los libros llamando a guardarLibros, por los tanto la complejidad teporal vuelve a ser O(n) con n cantidad de punteros a libros en el conjunto parcial.

En lo que refiere a uso de otras estructuras se podría decir que en lugar de usar un tda libro podríamos haber usado una estructura con todos los campo que elte posee pero no se aprecia por los menos a simple vista una mejora de la complejidad temporal mas bien uno entraría en el pensamiento de que cuando la complejidad temporal es lo que se mide no hay cambio realmente perceptibles. En cuantos al índice que en nuestro caso fue un arbolindice quizá se podría encontrar alguna forma de adaptar una estructura para reducir la complejidad temporal, ahora se nos viene a la mente en lugar de un árbol que sea una lista vinculada(una estructura nodolista enlazada a otro nodolista etc), o también un conjuto de índices a listas de libros de ese genero, pero es mucha la complicación de manejar semejante estructura con punteros listas conjuntas involucrados de forma distinta a la planteada, también habría que tomas en cuanta como llevar a cabo operaciones del resultado parcial en la ultima alternativa aquí mencionada.